WebSphere Application Server

BPM executes on a WebSphere Application Server environment. By and large there isn't much that the users of the product have to know but this section of the book will list some of the more pertinent areas in case the need arises.

Profile Management

Profiles are configurations of WAS servers. In general, profiles are created with the GUI PMT (Profile Management Tool) utility. This provides a wizard driven configuration of the profile instances. Through PMT WAS servers of a variety of types may be created.

References – Profile Management

- DeveloperWorks - The Support Authority: Take the confusion (and errors) out of managing profiles for WebSphere Application Server - 2010-07-14

Command line profile management

A command called manageprofiles is available to manage profiles. This command has a number of options:

```
manageprofiles -listProfiles
```

Will list the available profiles.

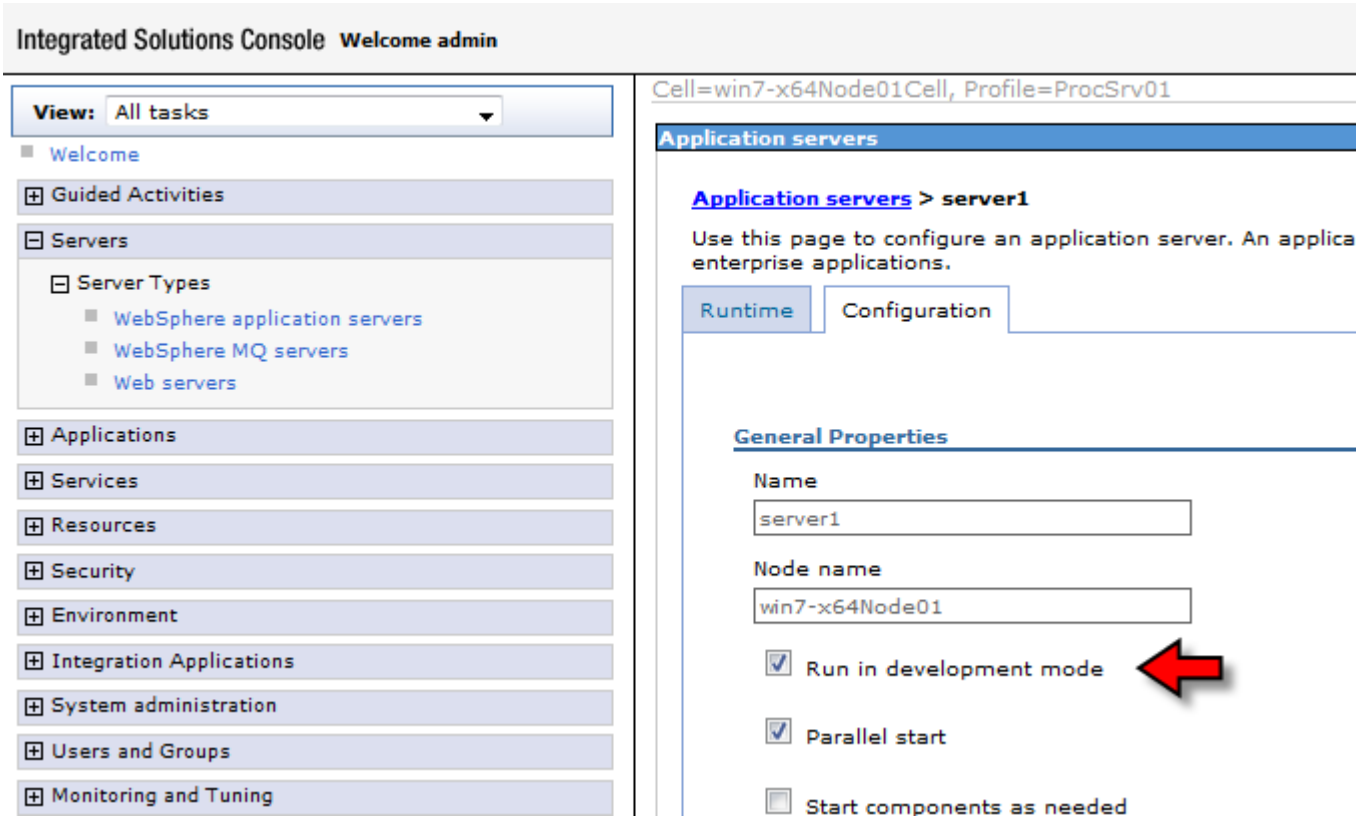
```
manageprofiles -delete -profileName *Name*
```

Will delete a named profile.

Switching a server from production to development mode

When profiles (including Process Center) are created, the servers are default defined to run in production mode. This is useful (naturally) for a production system but while developing solutions, this can get in your way. Running the server in development mode relaxes some constraints such as the ability to install a new application while there are existing instances of the old application already around.

From within the Admin console, there is a check box that specifies whether or not we run in development mode (default is **not** development).



Recovering from a failed adapter

install

If we have a module in ID that is directly deployed to PS that contains a JCA adapter which contains a configuration error and the module is deployed, the module will fail to deploy but will leave artifacts in the run-time that prevents **any** subsequent re-installs of that application. This is believed to be a bug and there is indeed plans for a fix. However, if it happens, here is a workaround:

- Stop the WAS server
- Delete files under
  - <WAS_HOME>/profiles/*profilename*/config/cells/*cellname*/cus/*<applicationName>*
  - <WAS_HOME>/profiles/*profilename*/config/cells/*cellname*/blas/*<applicationName>*
- Delete files under
  - <profile_root>/wstemp/
  - <profile_root>/temp/
  - <profile_root>/config/temp
  - <profile_root>/logs/dmgr
- Restart the server

Recovering from XA recovery

In rare circumstances, XA recovery may fail. To clean in-flight transactions and transaction logs, shutdown WAS and in the `<Profile>/tranlog/Cell/Node/server/transaction` directory, delete the contents of the folders called `partnerlog` and `tranlog`.

WAS Security

WAS Security is the topic of how the WAS server itself manages security artifacts.

See also:

- DeveloperWorks - WebSphere Application Server V7 advanced security hardening, Part 1: Overview and approach to security hardening – 2010-12-01
- DeveloperWorks - WebSphere Application Server V7 advanced security hardening, Part 2: Advanced security considerations – 2010-12-01

- DeveloperWorks - IBM WebSphere Developer Technical Journal: SSL, certificate, and key management enhancements for even stronger security in WebSphere Application Server V6.1 – 2006-12-06
- Redbook - WebSphere Application Server V7.0 Security Guide - 2011-04-25
- Redbook - IBM WebSphere Application Server V6.1 Security Handbook – 2006-12-28

Virtual Member Manager

The WebSphere Application Server (WAS) has a concept called the Virtual Member Manager. This is a security model that allows federated access to multiple authentication and user management systems. This is also known as the **federated user repository**.

Part of the concept of this is the notion of a *realm*. A realm is a domain of user identities. A VMM instance corresponds to a realm. Associated with a realm is a base entry which forms the root of a hierarchical structure for that realm.

Out of the box, WAS provides support for VMM to include three types of repository for users. These are:

- A file based repository
- One or more LDAP repositories
- A JDBC database repository

There are some restrictions when using VMM:

- The userids found in each repository must be unique
- If a single repository in the realm is down, all access is denied (by design)
- Only a single realm is supported

Using wsadmin, we can create user definitions using VMM. See AdminTask.createUser().

See also:

- DeveloperWorks - IBM WebSphere Developer Technical Journal: Expand your user registry options with a federated repository in WebSphere Application Server V6.1 – 2007-01-24
- DeveloperWorks - Sample virtual member manager custom adapter for WebSphere Application Server Version 6.1 – 2011-06

WebSphere Application Server WEB 2.0 Feature Pack

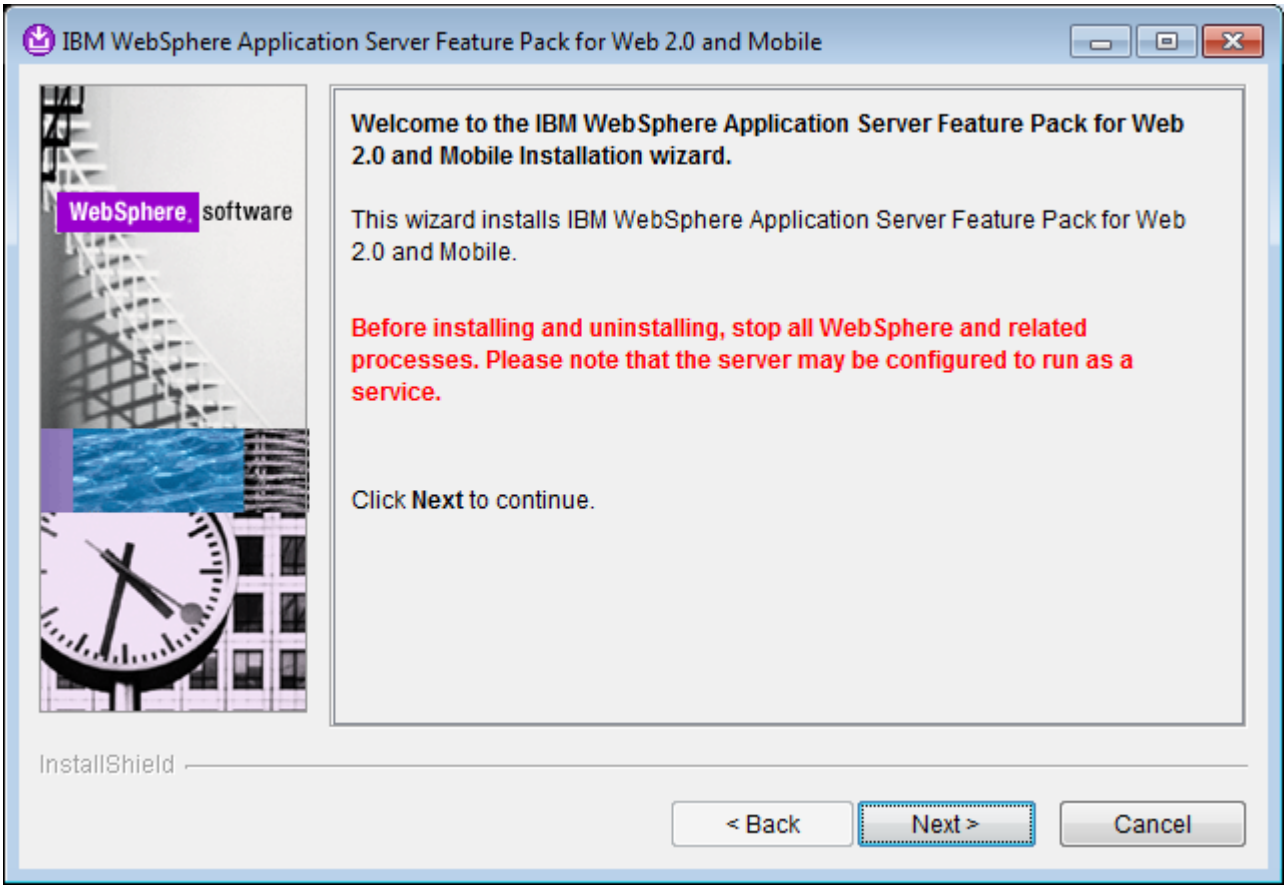
This feature pack is available for download here:

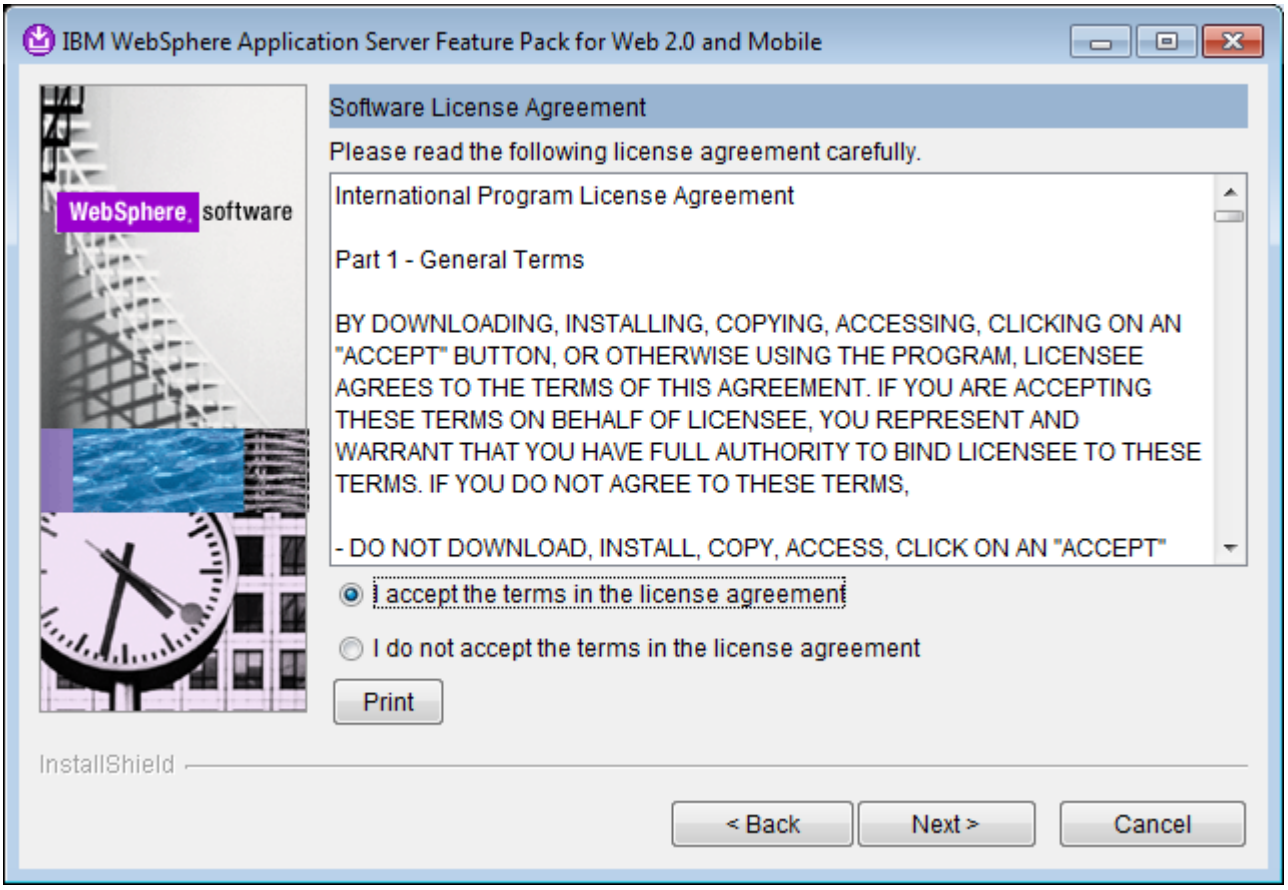
WebSphere Application Server Feature Pack for Web 2.0 and Mobile

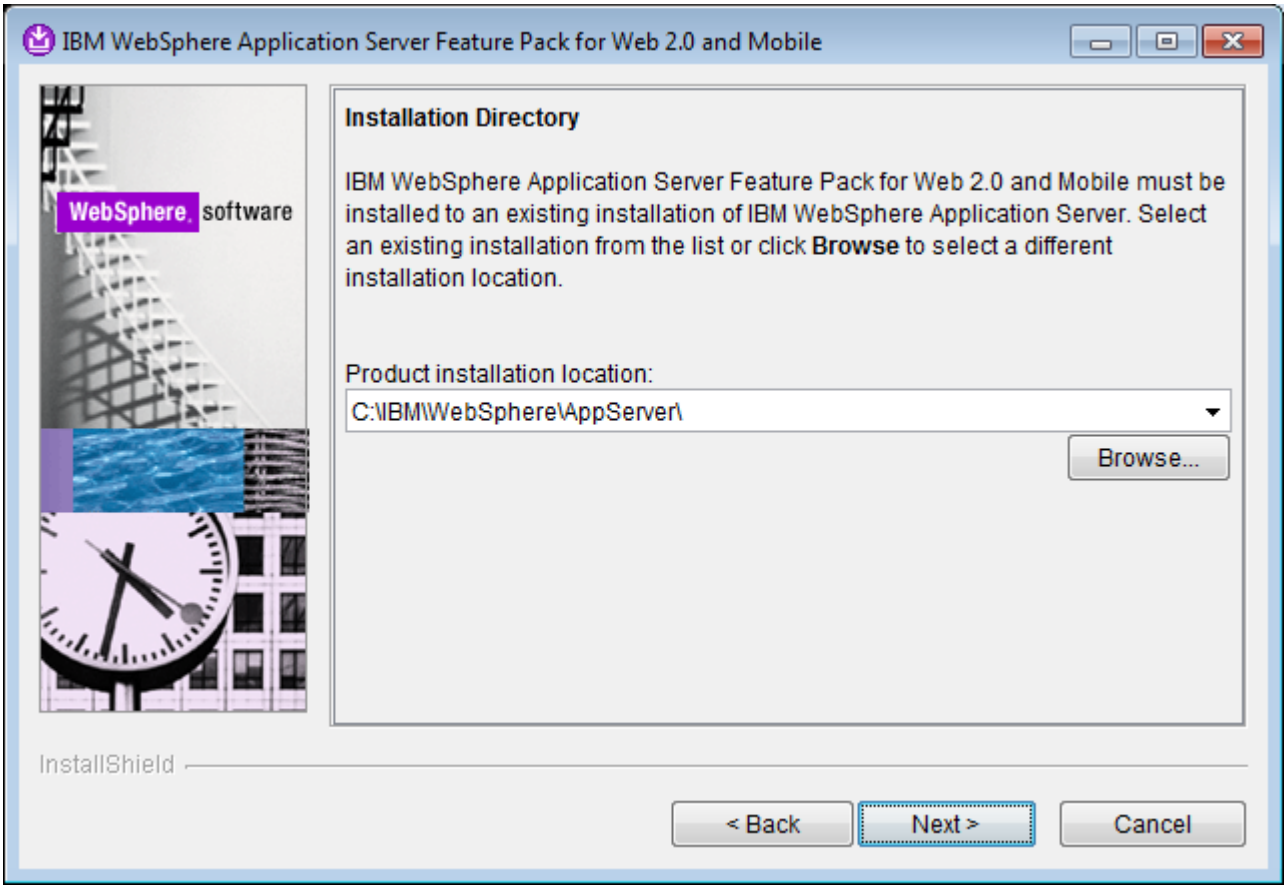
At the time of writing (03-2012) the version available was 1.1 and had a download size of about 264 Mbytes.

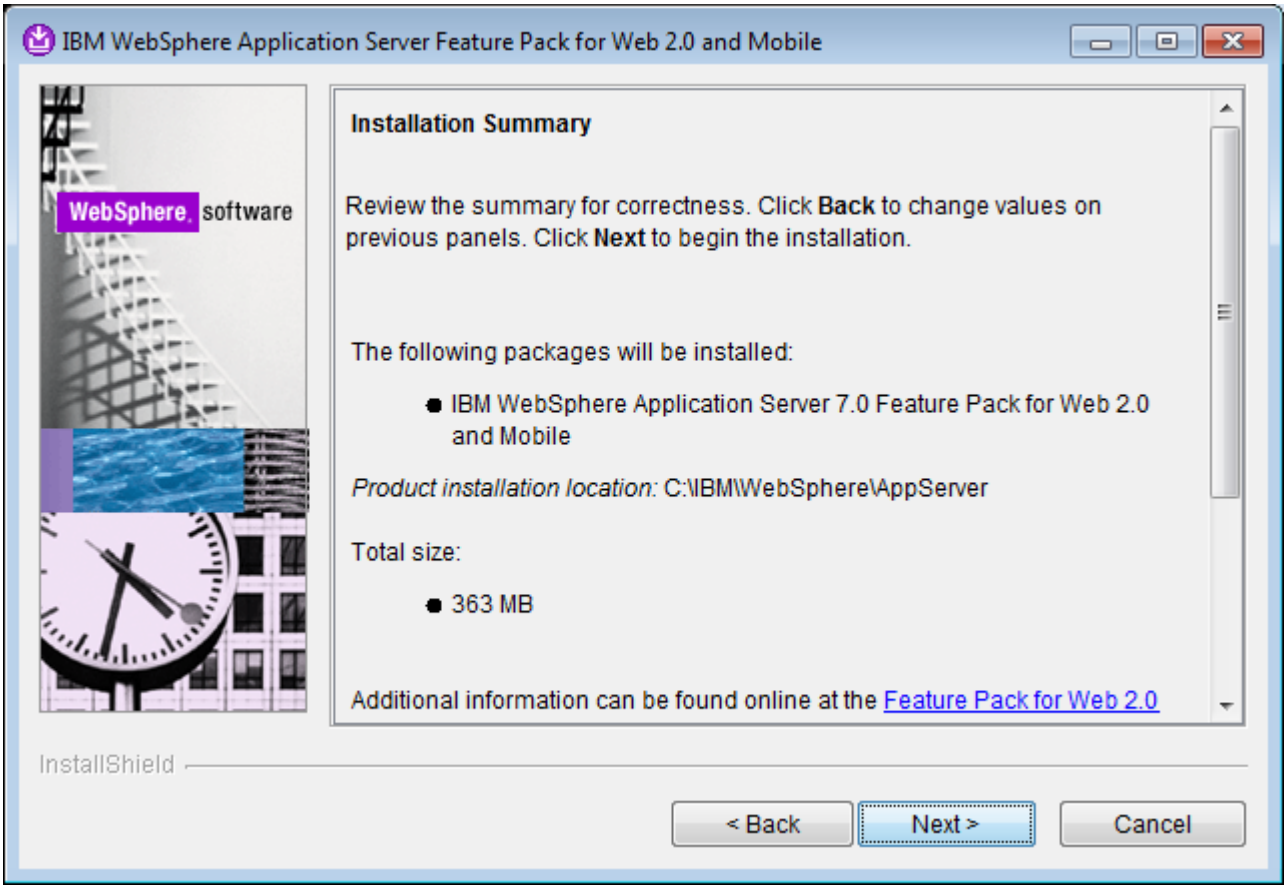
To install the feature pack, extract the ZIP and run the command:

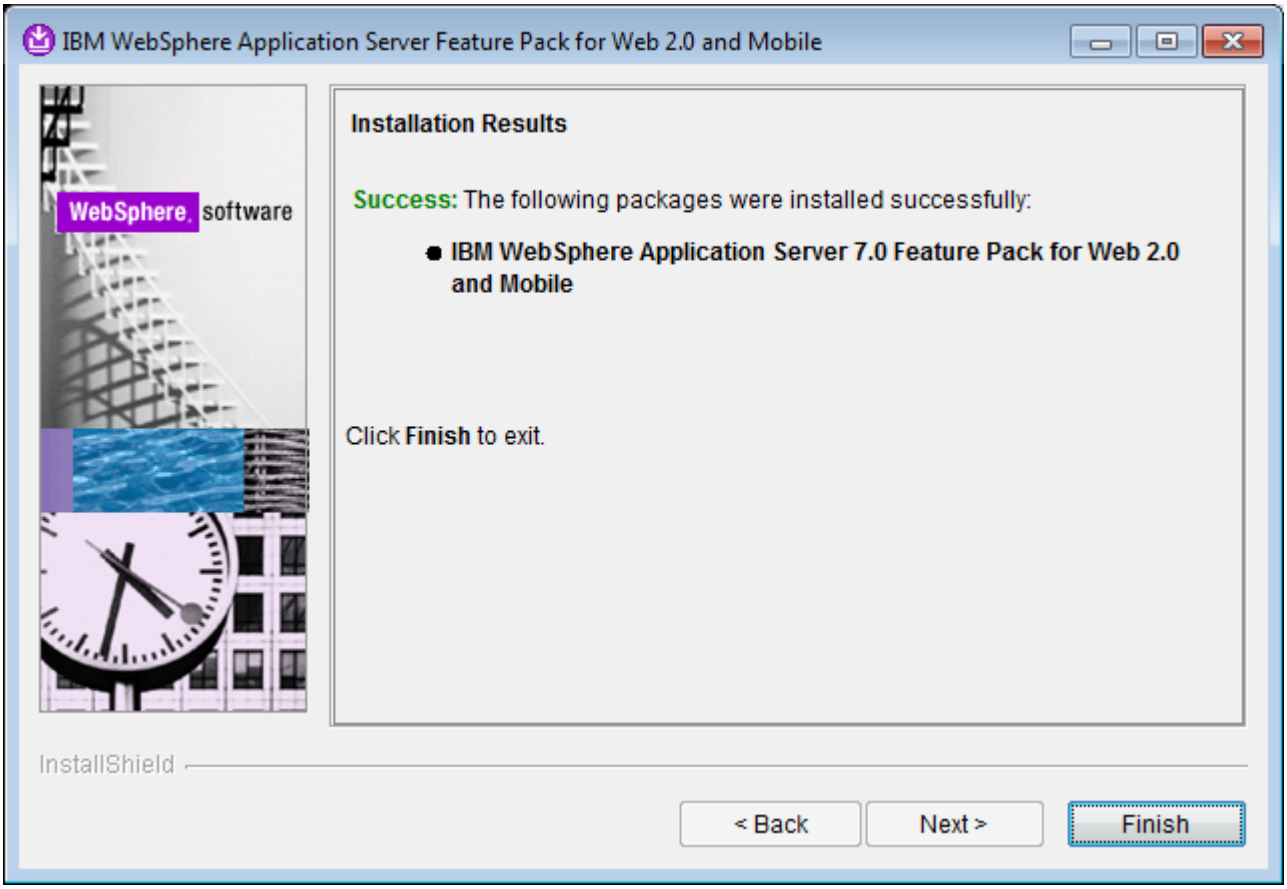
install -is:javahome <AppServer>\java



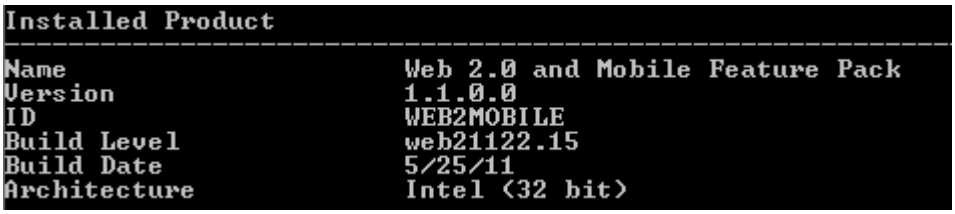

IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile
WebSphere. software
Software License Agreement
Please read the following license agreement carefully.
International Program License Agreement
Part 1 - General Terms
BY DOWNLOADING, INSTALLING, COPYING, ACCESSING, CLICKING ON AN "ACCEPT" BUTTON, OR OTHERWISE USING THE PROGRAM, LICENSEE AGREES TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCEPTING THESE TERMS ON BEHALF OF LICENSEE, YOU REPRESENT AND WARRANT THAT YOU HAVE FULL AUTHORITY TO BIND LICENSEE TO THESE TERMS. IF YOU DO NOT AGREE TO THESE TERMS,
- DO NOT DOWNLOAD, INSTALL, COPY, ACCESS, CLICK ON AN "ACCEPT"
I accept the terms in the license agreement
I do not accept the terms in the license agreement
Print
InstallShield
< Back
Next >
Cancel


IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile
WebSphere. software
Installation Directory
IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile must be installed to an existing installation of IBM WebSphere Application Server. Select an existing installation from the list or click Browse to select a different installation location.
Product installation location:
C:\IBM\WebSphere\AppServer\
Browse...
InstallShield
< Back
Next >
Cancel


IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile
WebSphere. software
Installation Summary
Review the summary for correctness. Click Back to change values on previous panels. Click Next to begin the installation.
The following packages will be installed:
IBM WebSphere Application Server 7.0 Feature Pack for Web 2.0 and Mobile
Product installation location: C:\IBM\WebSphere\AppServer
Total size:
363 MB
Additional information can be found online at the Feature Pack for Web 2.0
InstallShield
< Back
Next >
Cancel


IBM WebSphere Application Server Feature Pack for Web 2.0 and Mobile
WebSphere. software
Installation Results
Success: The following packages were installed successfully:
IBM WebSphere Application Server 7.0 Feature Pack for Web 2.0 and Mobile
Click Finish to exit.
InstallShield
< Back
Next >
Finish

Following a successful installation, running the WAS versionInfo command will show that the component is installed:



See Also:

- RedBook - Building Dynamic Ajax Applications Using WebSphere Feature Pack for Web 2.0 - SG24-7635-00

AJAX Proxy

Another feature provided as part of the Web 2.0 Feature Pack is known as the AJAX Proxy. This is a reverse proxy technology that allows a client to pass a service request through the proxy that will then be forwarded onwards to other servers. This effectively allows a web client to connect out to controlled external servers bypassing the sandbox rule of the browser connections.

For full details of the AJAX proxy, consult the InfoCenter. Here we will describe using the AJAX proxy as a stand-alone deployed application.

The AJAX proxy is governed by a control file with the name the `proxy-config.xml`. Within IBM BPM, this file can be found in:

`<Root>/BusinessSpace/mm.runtime/config`

To route through the proxy, target:

`http://<hostname>:<port>/mum/proxy/<protocol>/<host>:<port>/<url>`

For example:

`http://localhost:9081/mum/proxy/http/www.neilkolban.com`

By default, the proxy configuration is "locked down" to prevent access to the majority of target servers. One can modify the file and update the configuration.

```
AdminTask.updateBlobConfig('[-clusterName " <Cluster Name>" -propertyFileName "<Path\>/proxy-config.xml" -prefix "Mashups_"]')
AdminConfig.save()
```

After running this command, it will report:

`updateBlobConfig is executed succesfully`

Here is a concrete example.

Imagine my BPM server is running on `localhost:9081`. Imagine my Business Monitor server is running on `localhost:9085`. My browser, which is served from pages from BPM, wishes to make REST calls to Monitor ... how can this be achieved?

Here is an example REST call:

`http://localhost:9085/rest/bpm/monitor/models`

This needs to be made from a web page loaded from `localhost:9081`.

By using the Ajax proxy, we will end up sending the REST request to:

`http://localhost:9081/mum/proxy/http/localhost:9085/rest/bpm/monitor/models`

In the proxy-config.xml, we added the following:

```
<proxy:policy url="http://localhost:9085/*" acf="none" basic-auth-support="true">
<proxy:actions>
   <proxy:method\>GET\</proxy:method>
 </proxy:actions>
</proxy:policy>
```

Then the `updateBlobConfig()` command was run and the nodes synchronized. Now the request to access the monitor models works.

This is an XML file with a number of types of entries.

See also:

- TechNote: Troubleshooting IBM Business Process Manager: "403 BMWPX0006E: The URL you tried to access through the proxy is not allowed" – 2011-12-12

- Testing the AJAX Proxy in WebSphere Application Server 8.0 - 2013-01-22

JAX-RS – REST Services in Java

REST based services have become more and more common. The implementation of a REST based service provider in Java was originally built upon the notion of Java servlets but as REST became more pervasive, a better solution was proposed. This became known as JAX-RS and is documented in the Java specification as JSR-311.

This section provides a primer and notes on JAX-RS and, where relevant, how they might be leveraged with IBPM.

To start, we will examine some of the core concepts of the JAX-RS standard. Firstly, JAX-RS is based on Java annotations meaning it will only work in the later releases of Java.

We will start with the notion of the Root resources. A Root resource is a class that is associated with the annotation of "`@Path`".

For example:

```
@Path(value=”/myPath”)
public class MyClass
{
}
```

It appears that an annotation of:

`@Path(value="/myPath")`

is equivalent to:

`@Path("/myPath")`

Methods contained withing such a class can be flagged as the handlers for various HTTP requests. These are termed *resource methods*. For example:

```
@Path("/myPath")
public class MyClass
```

```
{
   @GET
   public String myFunc()
   {
      ...
   }
}
```

Signifies that if an HTTP GET request were made to “`/myPath`” then the Java method called “`myFunc()`” would be invoked.

An additional feature known as *sub-resource methods* can be defined that looks as follows:

```
@Path("/myPath")
public class MyClass
{
   @GET
   public String myFunc()
   {
     ...
   }
   @GET
   @Path("/myPart")
   public String my2ndFunc()
   {
     ...
   }
}
```

In this case, an HTTP request to “`/myPath/myPart`” would cause the function called “`my2ndFunc()`” to be invoked.

One further qualifier is provided that looks like:

```
@Path("/myPath")
public class MyClass
```

```
{
   @GET
   public String myFunc()
   {
      ...
   }
   @GET
   @Path("/myPart")
   public String my2ndFunc()
   {
      ...
   }
   @Path("/{myParam}/my3rdFunc")   public String my3rdFunc(@PathParam("myParam") String
myParam)
   {
      ...
   }
 }
```

Using this style, we can access the "`my3rdFunc()`" function using "`/myPath/someParam/my3rdFunc`". This style is known as a Sub-resource locater.

Another style of URL is known as matrix param:

```
@Path("/myPath")
public class MyClass
{
   @GET
   public String myFunc(@MatrixParam("fieldA") String fieldA,   @MatrixParam("fieldB") int
fieldB)
   {
      ...
   }
}
```

would handle requests such as:

`/myPath;fieldA=xyz;fieldB=123`

Similar to the above, query parameters can also be supplied:

```
@Path("/myPath")
public class MyClass
{
   @GET
   public String myFunc(
   @QueryParam("fieldA") String fieldA,
   @QueryParam("fieldB") int fieldB)
   {
      ...
   }
}
```

which would handle requests such as:

`/myPath?fieldA=hello&fieldB=123`

For method annotations, the following may be supplied corresponding to the HTTP commands:

- @GET
- @POST
- @PUT
- @DELETE
- @HEAD

HTTP Header parameters can be obtained by name in a method signature:

`public String myFunc(@HeaderParam("User-Agent") String userAgent, …)`

When invoked, the value of the "`User-Agent`" HTTP Header will be passed in as the `userAgent` parameter.

To access all headers, the following can be used:

`public String myFunc(@Context HttpHeader headers, …)`

When returning data, a return type of `javax.ws.rs.core.Response` can be used to set status code as well as HTTP Headers.

A request provides an indication of the data being passed into it. This is passed in with the HTTP

header known as “`Content-Type`”. A matching method can be declared using the `@Consumes` annotation:

```
@POST
@Consumes(value="text/xml")
public String doSomething()
{
   ...
}
```

Correspondingly, a method that returns a specific style of data can be declared to produce such with the `@Produces` annotation. An HTTP request with an HTTP header of "`Accept`" will be used to match the function.

```
@GET
@Produces(value="text/xml")
public String doSomethingElse()
{
   ...
}
```

Once the classes that are to be exposed as REST resources have been defined, they need to be “packaged” into a group associated with the an application that actually exposes them.

```
public class MyApplication extends javax.ws.rs.core.Application {   public Set<Class<?>>
getClasses()
   {
      Set<Class<?>> classes = new HashSet<Class<?>>();
classes.add(MyClassA.class);
      classes.add(MyClassB.class);
      return classes;
   }
 }
```

What we have here is a Java class that extends the JAX-RS Application class. This class overrides the method called "`getClasses()`" which returns a set of classes that are to be exposed as JAX-RS resources. In the example above, two classes are exposed.

The following maps the packages for various annotations:

Annotation	Package
`@Path`	`javax.ws.rs.Path`
`@PathParam`	`javax.ws.rs.PathParam`
`@HeaderParam`	`javax.ws.rs.HeaderParam`
`@Context`	`javax.ws.rs.core.Context`
`@MatrixParam`	`javax.ws.rs.MatrixParam`
`@QueryParam`	`javax.ws.rs.QueryParam`
`@CookieParam`	`javax.ws.rs.CookieParam`
`@FormParam`	`javax.ws.rs.FormParam`

See also:

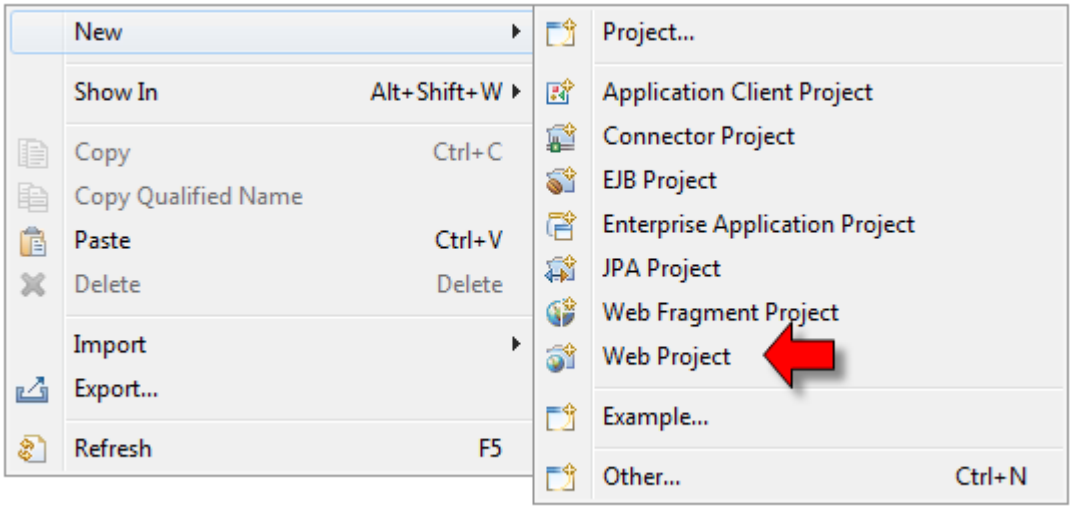
- WAS 8.5 InfoCenter – Overview of IBM JAX-RS
- RedBook - Building Dynamic Ajax Applications Using WebSphere Feature Pack for Web 2.0 - SG24-7635-00
- Wikipedia on JAX-RS
- JAX-RS: Developing RESTful Web Services in Java
- DeveloperWorks - Developing JAX-RS 1.1 RESTful Services in Rational Software Architect V8 for deployment to WebSphere Application Server V8 – 2011-11-02
- DeveloperWorks - Develop an Apache HttpClient client for Android to a JAX-RS web service – 2011-10-11
- DeveloperWorks - Build RESTful web services with Java technology – 2011-10-07
- DeveloperWorks - Comment lines: Combining Dojo and JAX-RS to create a RESTful service – 2010-07-14
- DeveloperWorks - Create RESTful Web services with Java technology – 2010-02-23
- DeveloperWorks - RESTful Web services: The basics – 2008-11-06

JAX RS in WebSphere Application Server

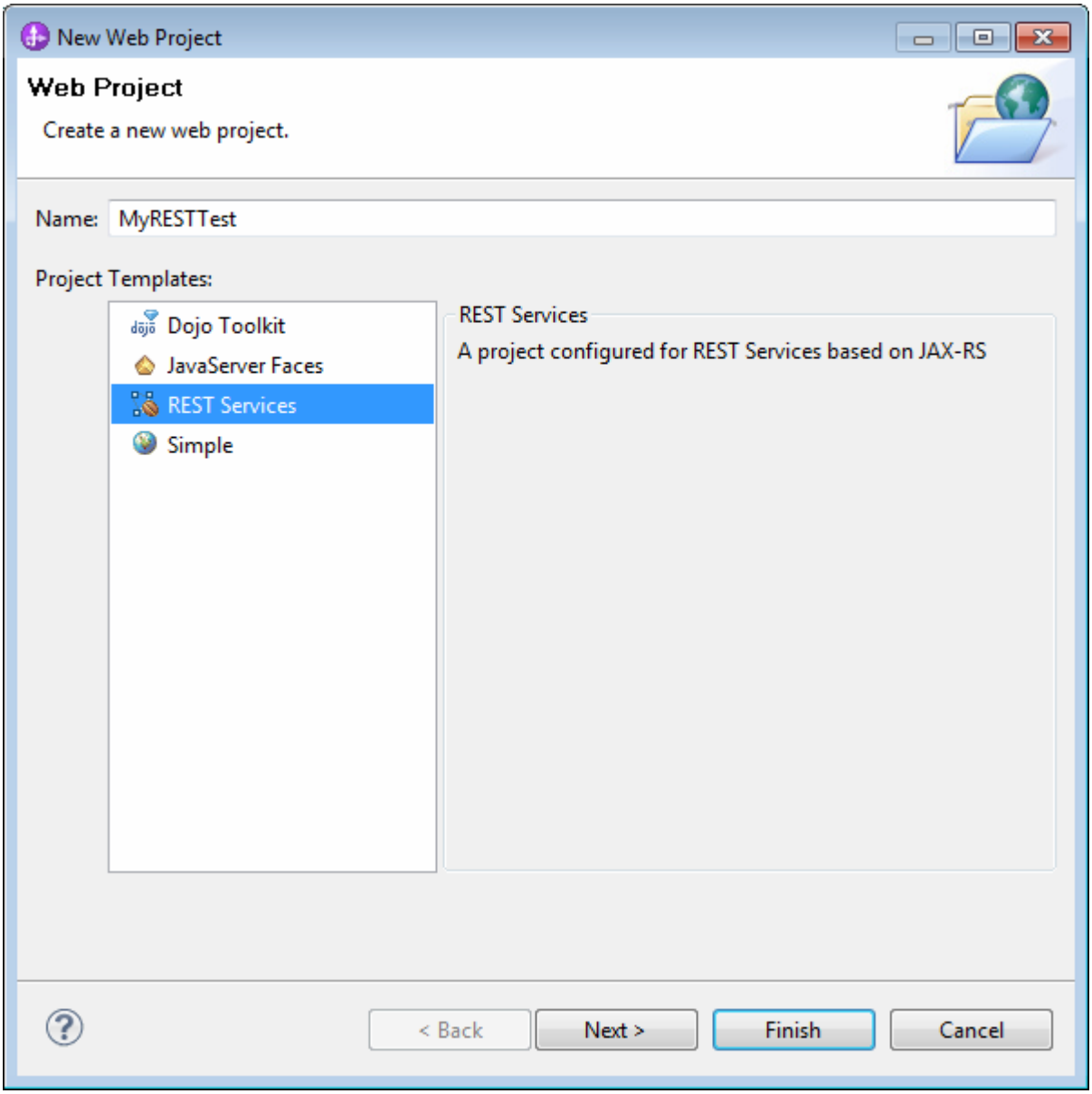
IBM WebSphere Application Server (WAS) provides an implementation of JAX-RS as part of the Web 2.0 Feature Pack and also as part of the base WAS 8.5 environment.

Here is a sample of building a JAX-RS exposed service in IBM Integration Designer.

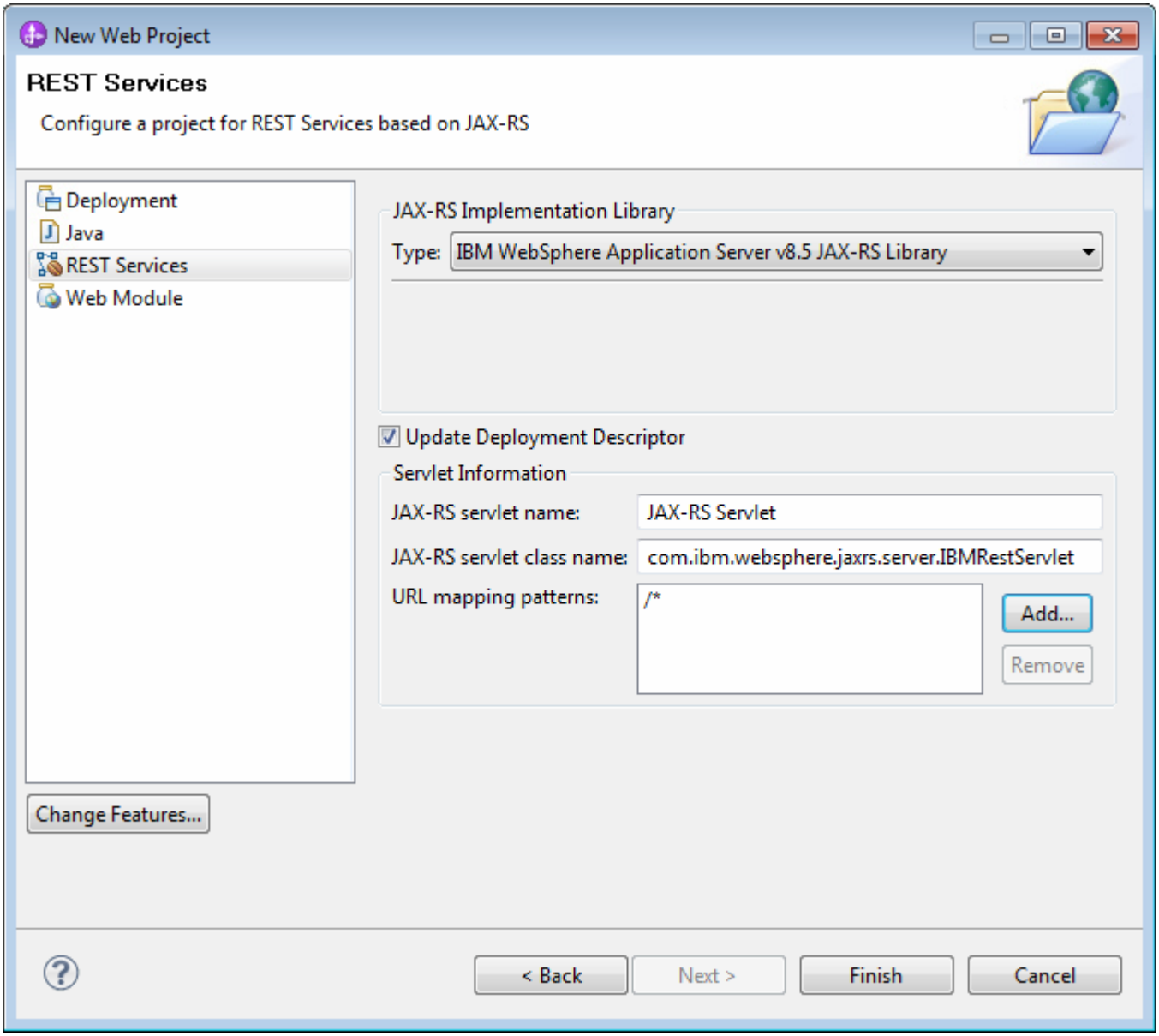
1. Create a new Web Project in IID. I called my project "MyRESTTest".



2. In the project settings, select "REST Services" as a template for project construction.



3. Change the URL mapping patterns to be "/*"



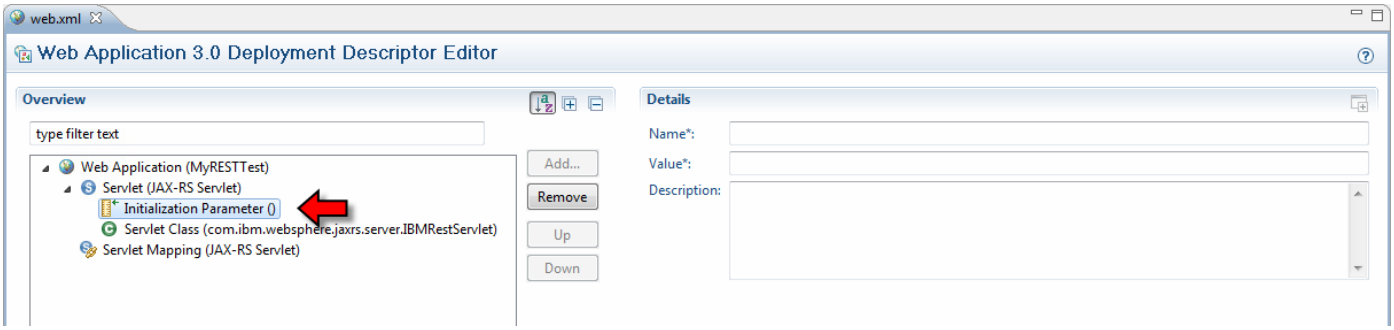
4. Create a Java class that implements a JAX-RS exposed service. This is the Java class that contains the code that will be invoked when the incoming REST request arrive. Notice that the class is annotated with the JAX-RS annotations.

Here is an example:

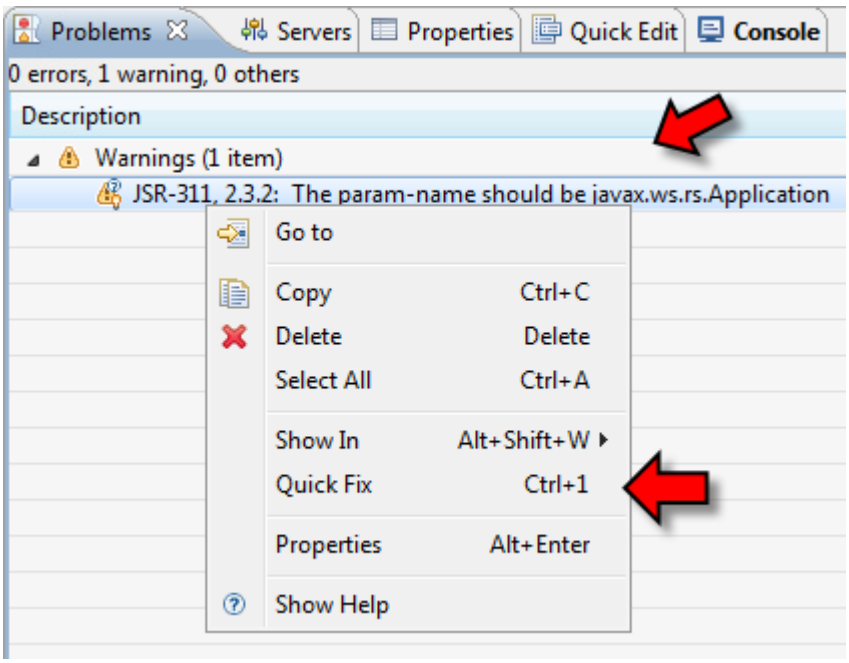
```
package com.ibm.demo;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
@Path("/rating")
public class Rating {
   @GET
```

```
    @Path("/id")
    public String getId() {
       System.out.println("/rating/id has been called!");
       return "Hi from /rating/id";
    }
}
```

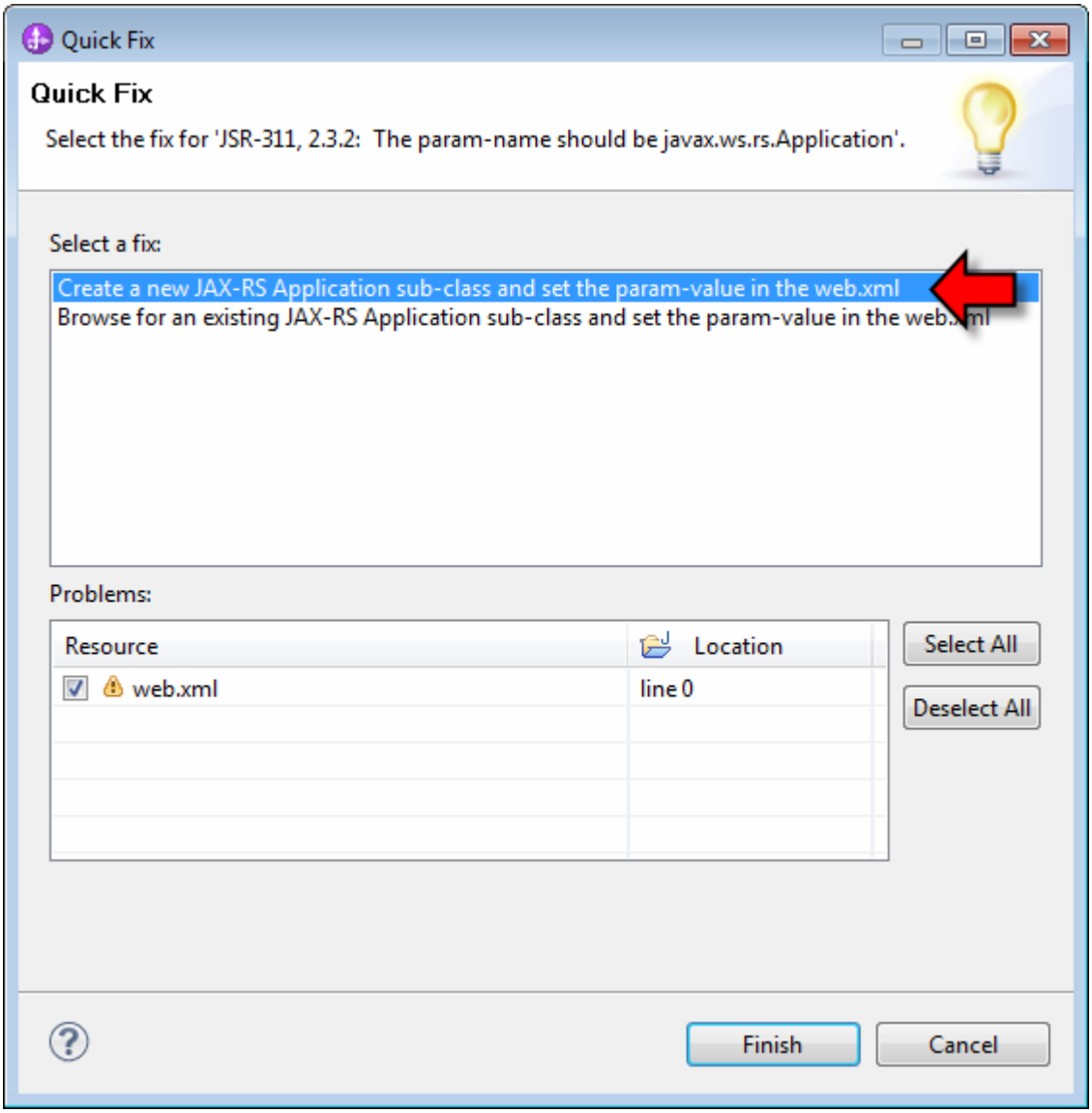
5. Open the Web deployment descriptor file called "web.xml". Add an empty "Initialization Parameter" and save the file. There is no question that this feels like an odd instruction. We could explicitly have added the values we need to the parameters or have had a better way of adding the details, but it "is what it is".



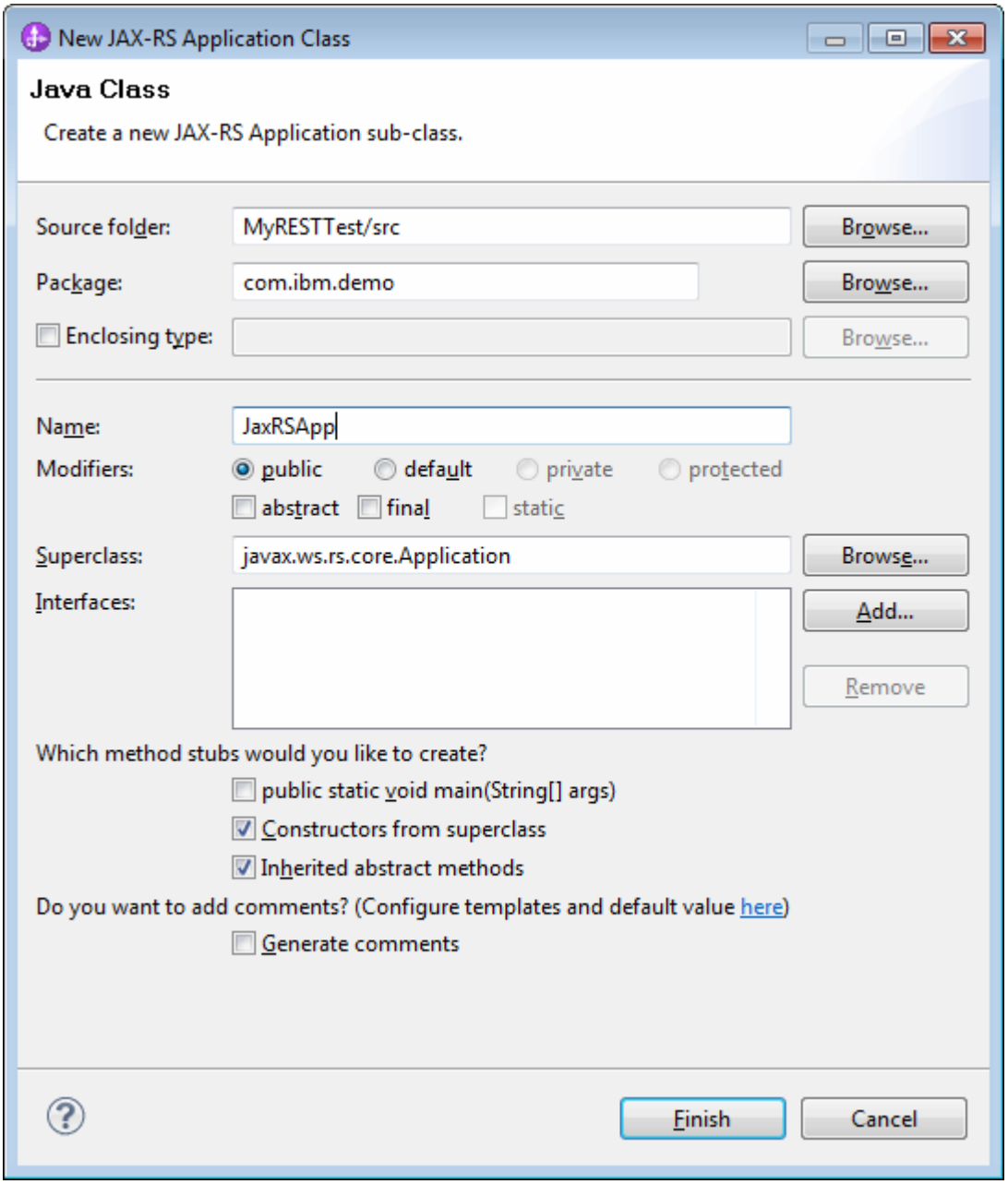
6. Execute the "Quick Fix" wizard with the "JSR-311 2.3.2" warning selected.



7. In the resulting wizard, select the option to create a new JAX-RS Application class.



8. Define the new class name and package to be created. In my example I called the Application class "JaxRSApp".



9. Implement the body of the newly created class. The Java source file generated is the "bootstrap" for our JAX-RS solution. We need to override a method called "getClasses" to return a Java Class reference to each of the implementations of JAX-RS function.

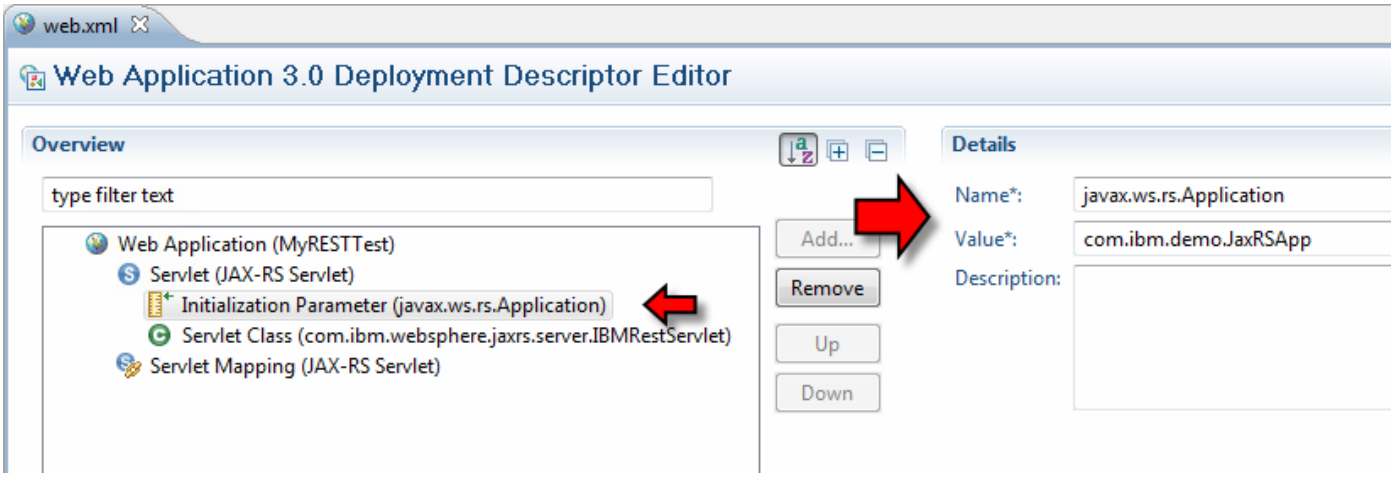
In my example, I coded:

```
package com.ibm.demo;
import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.core.Application;
```

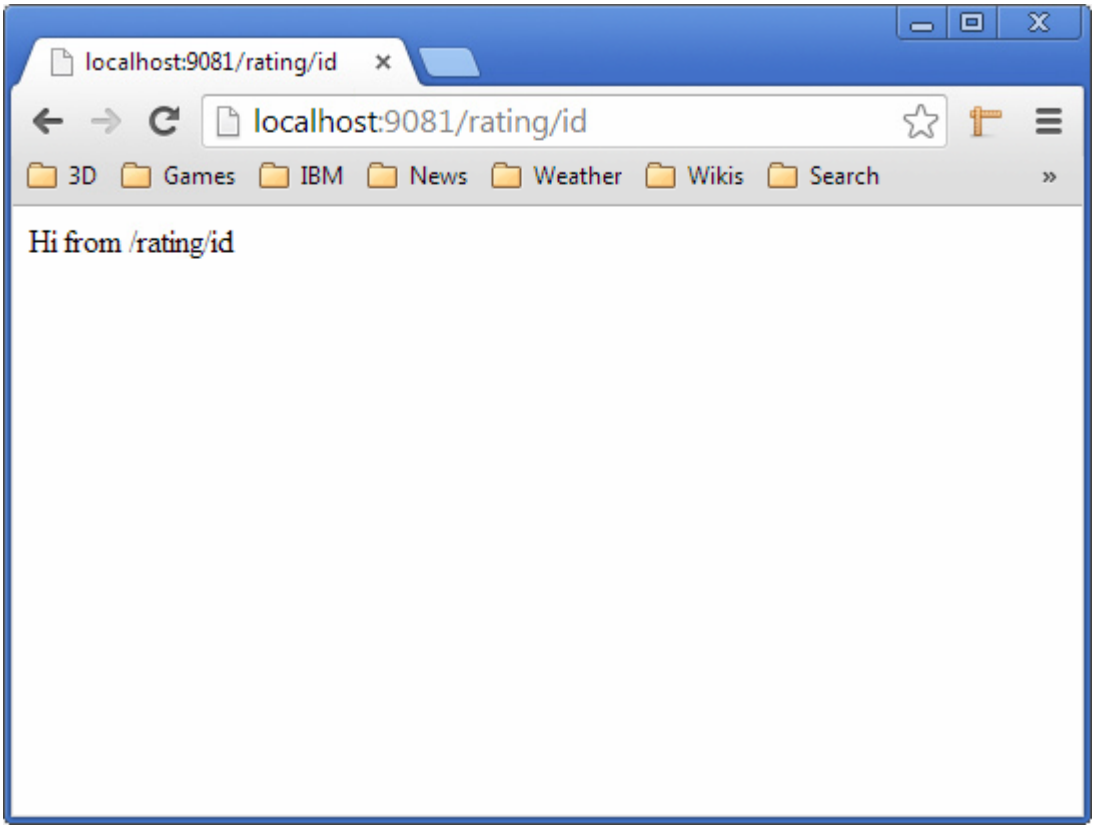
```
public class JaxRSApp extends Application {
   @Override
   public Set<Class<?>> getClasses() {      Set<Class<?>> classes = new
HashSet<Class<?>>();
      classes.add(Rating.class);
      return classes;
   }
}
```

Note that the class called "`Rating.class`" is the name of my class which contains the implementation of the exposed JAX-RS REST service.

10. (Optional) Examine the web.xml deployment description and see that it now points to our App as the loader for the JAX-RS environment.



11. Deploy the application to a suitable WAS server.
12. Test the REST service. We can use either a browser or a tool such as soapUI.



JAX-RS Clients

Not only does JAX-RS provide the ability to expose Java function as a REST accessible service, it also provides the capability to call A REST service from Java.

The package we will look at is called `javax.ws.rs.client`.

Unfortunately, this is simply not yet available in IBM BPM 8.5.5.

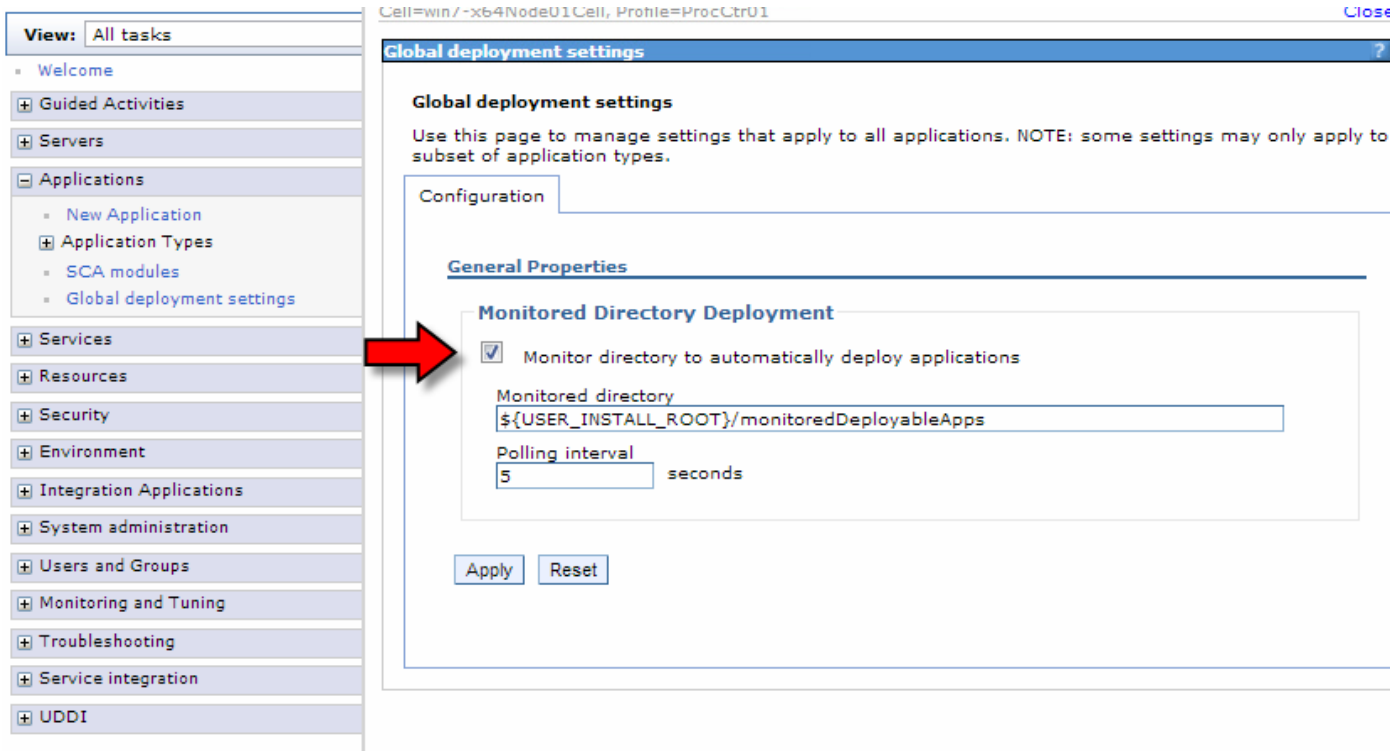
See also:

- Jersey – Client API

Deploying Java EE applications to WAS

There are times when we may wish to deploy Java EE applications to the WAS server hosting

Process Server. We can use the WAS admin console to achieve this task but there is an intriguing alternative. WAS provides a monitored directory concept. If a Java EE application is placed in that directory, WAS installs that application for us. By default, this capability is disabled and must be switched on. From the WAS admin console, switch to `Applications > Global deployment settings` and switch it on:



After enabling this property, the server must be restarted.

For stand-alone servers, the default location for the monitored directory is:

`<profileRoot>/monitoredDeployableApps/servers/server1`

WAS Messaging

- Redbook - WebSphere Application Server V7 Messaging Administration Guide

Foreign Connections

There are times when we wish to setup foreign connections between two SI Buses. Here are some notes.

On Machine B we have a Bus called "MONITOR.win7-x64Node01Cell.Bus". This has a queue defined on it called "TargetQ". We create a JMS Queue Connection Factor called "jms/Bus2QCF" that points to the Bus. We also create a JMS Queue Definition called "jms/TargetQ" that points to the "TargetQ" owned by SI Bus.

Property	Bus 1	Bus 2
BOOTSTRAP_ADDRESS	9810	2814
SIB_ENDPOINT_ADDRESS	7278	7281

Hermes Settings

	Bus 1	Bus 2
Session Name	Bus1	Bus2
binding	jms/Bus1QCF	jms/Bus2QCF
initialContextFactory	com.ibm.websphere.naming.WsnInitialContextFactory	com.ibm.websphere.naming.WsnInitialContextFact
providerURL	iiop://localhost:9810	iiop://localhost:2814

localhost:7278:BootstrapBasicMessaging

References

See also:

- Knowledge Center - WAS 8.5.5

Revision #3
Created 4 years ago by Admin
Updated 3 years ago by Neil Kolban